Vulkan

a.k.a. glNext

Did the future just arrive?
(Spring 2006)

”Next-generation OpenGL”?

What? When? Where?

New API, development started in 2014, released 16th february 2016.

Open specification - don’t expect it to hit all platforms immediately!

Big players try their own solutions. Apple has "Metal" instead. MS works though DirectX.

Background - AMD Mantle

Low-level API for more direct control of the GPU.

$$\begin{tikzcd}
\text{Mantle} \arrow[rr] \arrow[d] \arrow[dr, "\text{"Some features"?}"] & & \text{Vulkan, based on Mantle} \\
\text{Direct3D} & \text{Metal} &
\end{tikzcd}$$

So what is the point to us?

Promises:

- **Lower driver overhead**
- **More multi-thread friendly than OpenGL**
- **Shaders can be compiled to intermediary binary format (SPIR-V)**
- **Open front-end for shader compilers?**

Show me some code!

This is how you draw a triangle:

```c
#ifndef _MSC_VER
#define _ISOC11_SOURCE /* for aligned_alloc() */
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <assert.h>

#ifdef _WIN32
#pragma comment(linker, "/subsystem:windows")
#define APP_NAME_STR_LEN 80
#endif // _WIN32

#include <vulkan/vulkan.h>

#define DEMO_TEXTURE_COUNT 1
#define VERTEX_BUFFER_BIND_ID 0
#define APP_SHORT_NAME "tri"
#define APP_LONG_NAME "The Vulkan Triangle Demo Program"

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))

#if defined(NDEBUG) && defined(__GNUC__)
#define U_ASSERT_ONLY __attribute__((unused))
#else
#define U_ASSERT_ONLY
#endif

#ifdef _WIN32
#define ERR_EXIT(err_msg, err_class)                                    \
    do {                                                                \
        MessageBox(NULL, err_msg, err_class, MB_OK);                   \
        exit(1);                                                        \
    } while (0)
#else // _WIN32

#define ERR_EXIT(err_msg, err_class)                                    \
    do {                                                                \
        printf(err_msg);                                                \
        fflush(stdout);                                                 \
        exit(1);                                                        \
    } while (0)
#endif // _WIN32

#define GET_INSTANCE_PROC_ADDR(inst, entrypoint)                        \
    {                                                                   \
        demo->fp##entrypoint =                                          \
            (PFN_vk##entrypoint)vkGetInstanceProcAddr(inst, "vk" #entrypoint); \
        if (demo->fp##entrypoint == NULL) {                             \
            ERR_EXIT("vkGetInstanceProcAddr failed to find vk" #entrypoint,   \
                     "vkGetInstanceProcAddr Failure");                  \
        }                                                               \
    }

#define GET_DEVICE_PROC_ADDR(dev, entrypoint)                           \
    {                                                                   \
        demo->fp##entrypoint =                                          \
            (PFN_vk##entrypoint)vkGetDeviceProcAddr(dev, "vk" #entrypoint);   \
        if (demo->fp##entrypoint == NULL) {                             \
            ERR_EXIT("vkGetDeviceProcAddr failed to find vk" #entrypoint,     \
                     "vkGetDeviceProcAddr Failure");                    \
        }                                                               \
    }

struct texture_object {
    VkSampler sampler;

    VkImage image;
    VkImageLayout imageLayout;

    VkDeviceMemory mem;
    VkImageView view;
    int32_t tex_width, tex_height;
};

VKAPI_ATTR VkBool32 VKAPI_CALL
dbgFunc(VkFlags msgFlags, VkDebugReportObjectTypeEXT objType,
        uint64_t srcObject, size_t location, int32_t msgCode,
        const char *pLayerPrefix, const char *pMsg, void *pUserData) {
    char *message = (char *)malloc(strlen(pMsg) + 100);

    assert(message);

    if (msgFlags & VK_DEBUG_REPORT_ERROR_BIT_EXT) {
        sprintf(message, "ERROR: [%s] Code %d : %s", pLayerPrefix, msgCode,
                pMsg);
    } else if (msgFlags & VK_DEBUG_REPORT_WARNING_BIT_EXT) {
        sprintf(message, "WARNING: [%s] Code %d : %s", pLayerPrefix, msgCode,
                pMsg);
    } else {
        return false;
    }

#ifdef _WIN32
    MessageBox(NULL, message, "Alert", MB_OK);
#else
    printf("%s\n", message);
    fflush(stdout);
#endif
    free(message);

    /*
     * false indicates that layer should not bail-out of an
     * API call that had validation failures. This may mean that the
     * app dies inside the driver due to invalid parameter(s).
     * That's what would happen without validation layers, so we'll
     * keep that behavior here.
     */
    return false;
}

typedef struct _SwapchainBuffers {
    VkImage image;
    VkCommandBuffer cmd;
    VkImageView view;
} SwapchainBuffers;

struct demo {
#ifdef _WIN32
#define APP_NAME_STR_LEN 80
    HINSTANCE connection;        // hInstance - Windows Instance
    char name[APP_NAME_STR_LEN]; // Name to put on the window/icon
    HWND window;                 // hWnd - window handle
#else                            // _WIN32
    xcb_connection_t *connection;
    xcb_screen_t *screen;
    xcb_window_t window;
    xcb_intern_atom_reply_t *atom_wm_delete_window;
#endif                           // _WIN32
    VkSurfaceKHR surface;
    bool prepared;
    bool use_staging_buffer;

    VkInstance inst;
    VkPhysicalDevice gpu;
    VkDevice device;
    VkQueue queue;
    VkPhysicalDeviceProperties gpu_props;
    VkQueueFamilyProperties *queue_props;
    uint32_t graphics_queue_node_index;

    uint32_t enabled_extension_count;
    uint32_t enabled_layer_count;
    char *extension_names[64];
    char *device_validation_layers[64];

    int width, height;
    VkFormat format;
    VkColorSpaceKHR color_space;

    PFN_vkGetPhysicalDeviceSurfaceSupportKHR
        fpGetPhysicalDeviceSurfaceSupportKHR;
    PFN_vkGetPhysicalDeviceSurfaceCapabilitiesKHR
        fpGetPhysicalDeviceSurfaceCapabilitiesKHR;
    PFN_vkGetPhysicalDeviceSurfaceFormatsKHR
        fpGetPhysicalDeviceSurfaceFormatsKHR;
    PFN_vkGetPhysicalDeviceSurfacePresentModesKHR
        fpGetPhysicalDeviceSurfacePresentModesKHR;
    PFN_vkCreateSwapchainKHR fpCreateSwapchainKHR;
    PFN_vkDestroySwapchainKHR fpDestroySwapchainKHR;
    PFN_vkGetSwapchainImagesKHR fpGetSwapchainImagesKHR;
    PFN_vkAcquireNextImageKHR fpAcquireNextImageKHR;
    PFN_vkQueuePresentKHR fpQueuePresentKHR;
    uint32_t swapchainImageCount;
    VkSwapchainKHR swapchain;
    SwapchainBuffers *buffers;

    VkCommandPool cmd_pool;

    struct {
        VkFormat format;

        VkImage image;
        VkDeviceMemory mem;
        VkImageView view;
    } depth;

    struct texture_object textures[DEMO_TEXTURE_COUNT];

    struct {
        VkBuffer buf;
        VkDeviceMemory mem;

        VkPipelineVertexInputStateCreateInfo vi;
        VkVertexInputBindingDescription vi_bindings[1];
        VkVertexInputAttributeDescription vi_attrs[2];
    } vertices;

    VkCommandBuffer setup_cmd; // Command Buffer for initialization commands
    VkCommandBuffer draw_cmd;  // Command Buffer for drawing commands
    VkPipelineLayout pipeline_layout;
    VkDescriptorSetLayout desc_layout;
    VkPipelineCache pipelineCache;
    VkRenderPass render_pass;
    VkPipeline pipeline;

    VkShaderModule vert_shader_module;
    VkShaderModule frag_shader_module;

    VkDescriptorPool desc_pool;
    VkDescriptorSet desc_set;

    VkFramebuffer *framebuffers;

    VkPhysicalDeviceMemoryProperties memory_properties;

    bool validate;
    PFN_vkCreateDebugReportCallbackEXT CreateDebugReportCallback;
    PFN_vkDestroyDebugReportCallbackEXT DestroyDebugReportCallback;
    VkDebugReportCallbackEXT msg_callback;
    PFN_vkDebugReportMessageEXT DebugReportMessage;

    float depthStencil;
    float depthIncrement;

    bool quit;
    uint32_t current_buffer;
    uint32_t queue_count;
};

// Forward declaration:
static void demo_resize(struct demo *demo);

static bool memory_type_from_properties(struct demo *demo, uint32_t typeBits,
                                        VkFlags requirements_mask,
                                        uint32_t *typeIndex) {
    // Search memtypes to find first index with those properties
    for (uint32_t i = 0; i < 32; i++) {
        if ((typeBits & 1) == 1) {
            // Type is available, does it match user properties?
            if ((demo->memory_properties.memoryTypes[i].propertyFlags &
                 requirements_mask) == requirements_mask) {
                *typeIndex = i;
                return true;
            }
        }
        typeBits >>= 1;
    }
    // No memory types matched, return failure
    return false;
}

static void demo_flush_init_cmd(struct demo *demo) {
    VkResult U_ASSERT_ONLY err;

    if (demo->setup_cmd == VK_NULL_HANDLE)
        return;

    err = vkEndCommandBuffer(demo->setup_cmd);
    assert(!err);

    const VkCommandBuffer cmd_bufs[] = {demo->setup_cmd};
    VkFence nullFence = {VK_NULL_HANDLE};
    VkSubmitInfo submit_info = {.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO,
                                .pNext = NULL,
                                .waitSemaphoreCount = 0,
                                .pWaitSemaphores = NULL,
                                .pWaitDstStageMask = NULL,
                                .commandBufferCount = 1,
                                .pCommandBuffers = cmd_bufs,
                                .signalSemaphoreCount = 0,
                                .pSignalSemaphores = NULL};

    err = vkQueueSubmit(demo->queue, 1, &submit_info, nullFence);
    assert(!err);

    err = vkQueueWaitIdle(demo->queue);
    assert(!err);

    vkFreeCommandBuffers(demo->device, demo->cmd_pool, 1, cmd_bufs);
    demo->setup_cmd = VK_NULL_HANDLE;
}

static void demo_set_image_layout(struct demo *demo, VkImage image,
                                  VkImageAspectFlags aspectMask,
                                  VkImageLayout old_image_layout,
                                  VkImageLayout new_image_layout,
                                  VkAccessFlagBits srcAccessMask) {

    VkResult U_ASSERT_ONLY err;
```

These are the first ≠300 lines of a file with 2448 lines... should I continue?

What kind of mind-boggling bullsh*t is this?

- **Low-level**
- **Detailed control over buffers, processes, queues, devices...**

This comes for a cost!

Important: This does NOT mean that OpenGL will be discontinued in the foreseeable future! Vulkan is a lower-level complement, not a replacement!

Much is happening! Computer graphics is still moving fast...